UNITED STATES PATENT APPLICATION
FOR

# SELECTIVE EXPANSION OF HIGH-LEVEL DESIGN LANGUAGE MACROS FOR AUTOMATED DESIGN MODIFICATION

INVENTORS:

GREGORY HIBDON
a citizen of the United States, residing at
108 SEERPREEN WAY
FOLSOM, CALIFORNIA 95630

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026
(303) 740-1980

DOCKET: 5725.750

**EXPRESS MAIL CERTIFICATE OF MAILING**

"Express Mail" mailing label number: EL750127621US
Date of Deposit: December 30, 2000
I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to the Commissioner of Patents and Trademarks, Washington, D. C. 20231

Heather S. South
(Typed or printed name of person mailing paper or fee)

Heather S. South
(Signature of person mailing paper or fee)

12/30/00
(Date signed)

# SELECTIVE EXPANSION OF HIGH-LEVEL DESIGN LANGUAGE MACROS FOR AUTOMATED DESIGN MODIFICATION

## FIELD OF THE INVENTION

The invention relates generally to the field of design-for-test (DFT) and scan insertion. More particularly, the invention relates to selectively expanding macros within a hardware description language (HDL) in a context sensitive manner.

## BACKGROUND OF THE INVENTION

Complex Very Large Scale Integration (VLSI) designs such as microprocessors are typically designed in HDL. A design represented in such an HDL is then translated into circuitry to be etched onto silicon. A common practice during design is to insert test points to allow more efficient testing of the device. The inserted cells can be checked during testing to verify smaller circuits within the larger design and thereby provide a more effective and efficient means of testing the device. Scan insertion is a standard design-for-test (DFT) method that enables structural test generation in such a manner.

There are various standard software tools used by chip designers to insert scan cells for selected signals into the HDL design files comprising a microprocessor design. Currently, these scan insertion tools require HDL design files to be run through a preprocessing program similar to a C language preprocessor that handles compile time variable processing and expansion of HDL macro constructs prior to the scan insertion process. In addition to complexity and performance limitations, one of the significant disadvantages of current scan insertion tools is

that the resulting scan inserted HDL design files look very different from the chip designer's original design files due to the wholesale changes made by the preprocessor. Having design files that look so "foreign" to the designer makes verification and debugging of the designs much more difficult.

DFT insertion, such as insertion of scan cells, into HDL can be performed in different ways by the standard software tools. Some tools highlight the HDL segment that needs to be modified and use smart editors that understand HDL and the DFT context. Although such tools exist in the industry, this method still requires significant effort whenever DFT requirements change. Other tools preprocess the HDL and insert DFT into the preprocessed HDL. Although this method makes the insertion a simpler problem, the resulting HDL does not look similar to the original HDL design making it hard to debug.

0603.742.5959

## BRIEF DESCRIPTION OF THE DRAWINGS

The appended claims set forth the features of the invention with particularity. The invention, together with its advantages, may be best understood from the following detailed description taken in conjunction with the accompanying drawings of which:

**Figure 1** is a block diagram of a typical computer system upon which one embodiment of the present invention may be implemented;

**Figure 2** is a block diagram of a high level architectural view of selective expansion of HDL macros according to one embodiment of the present invention;

**Figure 3** is a flowchart of tokenizer module processing according to one embodiment of the present invention;

**Figure 4** is a flowchart of "as if" parser processing according to one embodiment of the present invention;

**Figure 5** is a block diagram of a token object according to one embodiment of the present invention;

**Figure 6** is a block diagram illustrating an example of the use of multifaceted tokens according to one embodiment of the present invention;

**Figure 7** is a block diagram illustrating results of tokenization as seen by an as if parser and an output module according to one embodiment of the present invention; and

**Figure 8** is a flow diagram illustrating file output module processing according to one embodiment of the present invention.

## DETAILED DESCRIPTION OF THE INVENTION

A method and apparatus are described for selective expansion of HDL macros for automated design modification. According to one embodiment of the present invention, the selective expansion of HDL macros allows for insertion of scan cells for selected signals into HDL design files representing a hardware design, such as the design of a microprocessor, while making the modified file look as much as possible like the chip designer's original HDL file by using an "as if" approach to parsing HDL design macros, multifaceted parser tokens, and a three-tiered token list. In order to make the modified file look as much as possible like the chip designer's original HDL file, all text except the required changes are preserved from the original file. To accomplish this, the parsing program used by the HDL scan insertion tool creates lists of tokens that record everything including spaces, tabs, and comments. Then to perform the modifications to the HDL, the token lists, representing "lines" from the HDL file, are modified before they are written back out as the updated scan inserted HDL file.

In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without some of these specific details. In other instances, well-known structures and devices are shown in block diagram form.

The present invention includes various steps, which will be described below. The steps of the present invention may be performed by hardware components or may be embodied in machine-executable instructions, which may be used to cause a general-purpose or special-purpose processor or logic circuits programmed with the instructions to perform the steps. Alternatively, the steps may be performed by a combination of hardware and software.

The present invention may be provided as a computer program product which may include a machine-readable medium having stored thereon instructions which may be used to program a computer (or other electronic devices) to perform a process according to the present invention. The machine-readable medium may include, but is not limited to, floppy diskettes, optical disks, CD-ROMs, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, magnetic or optical cards, flash memory, or other type of media / machine-readable medium suitable for storing electronic instructions. Moreover, the present invention may also be downloaded as a computer program product, wherein the program may be transferred from a remote computer to a requesting computer by way of data signals embodied in a carrier wave or other propagation medium via a communication link (e.g., a modem or network connection).

Importantly, while embodiments of the present invention will be described with reference to HDL, scan macros, and integrated circuit design tools, the method and apparatus described herein are thought to be equally applicable to other types of source code compilers and/or preprocessors.

Before describing selective expansion of HDL macros for automated design modification according to various embodiments of the present invention, some terms that will be used throughout this application will briefly be defined.

"As if parsing" generally refers to a process wherein characters within input data are separated in a manner that allows the output of the process to appear that any macros present in the input data have been expanded.

"HDL" stands for Hardware Description Language and means a language, including but not limited to Verilog, VHDL, or other high-level programming languages such as C, C++, etc., that can be used to design integrated circuits.

A "line" consists of one or more ASCII characters followed by a carriage return or other end-of-line symbol.

A "macro" is a name representing a series of commands or functions.

"Macro expansion" generally refers to the process of replacing a macro name with the series of commands or functions that the name represents.

"Scan insertion" generally refers to the process of adding observation points or "scan cells" to a hardware design representation. Typically, scan insertion takes place in a schematic representation. However, according to embodiments of the present invention, scan insertion may involve modification of an HDL file.

A "scan macro" is a name representing a series of scan commands.

A "token" is a single element of a line of data such as a keyword, operator, or punctuation mark.

According to one embodiment of the present invention, in order to make the modified file look as much as possible like the chip designer's original HDL file, all text except the required changes are preserved from the original file. To accomplish this, the parsing program used by the HDL scan insertion tool creates lists of tokens that record everything including spaces, tabs, and comments. Then to perform the modifications to the HDL, the token lists representing "lines" from the HDL file are modified, before they are written back out as the updated scan inserted HDL file.

Figure 1 is a block diagram of a typical computer system upon which one embodiment of the present invention may be implemented. In this example, computer system 100 comprises a bus or other communication means 101 for communicating information, and a processing means such as processor 102 coupled with bus 101 for processing information. Computer system 100 further comprises a random access memory (RAM) or other dynamic storage device 104 (referred to as

main memory), coupled to bus 101 for storing information and instructions to be executed by processor 102. Main memory 104 also may be used for storing temporary variables or other intermediate information during execution of instructions by processor 102. Computer system 100 also comprises a read only memory (ROM) and/or other static storage device 106 coupled to bus 101 for storing static information and instructions for processor 102.

A data storage device 107 such as a magnetic disk or optical disc and its corresponding drive may also be coupled to computer system 100 for storing information and instructions. Computer system 100 can also be coupled via bus 101 to a display device 121, such as a cathode ray tube (CRT) or Liquid Crystal Display (LCD), for displaying information to an end user. Typically, an alphanumeric input device 122, including alphanumeric and other keys, may be coupled to bus 101 for communicating information and/or command selections to processor 102. Another type of user input device is cursor control 123, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 102 and for controlling cursor movement on display 121.

A communication device 125 is also coupled to bus 101. The communication device 125 may include a modem, a network interface card, or other well-known interface devices, such as those used for coupling to Ethernet, token ring, or other types of physical attachment for purposes of providing a communication link to support a local or wide area network, for example. In this manner, the computer system 100 may be coupled to a number of clients and/or servers via a conventional network infrastructure, such as a company's Intranet and/or the Internet, for example.

It is appreciated that a lesser or more equipped computer system than the example described above may be desirable for certain implementations. Therefore, the configuration of computer system 100 will vary from implementation to implementation depending upon

numerous factors, such as price constraints, performance requirements, technological improvements, and/or other circumstances.

It should be noted that, while the steps described herein may be performed under the control of a programmed processor, such as processor 102, in alternative embodiments, the steps may be fully or partially implemented by any programmable or hardcoded logic, such as Field Programmable Gate Arrays (FPGAs), TTL logic, or Application Specific Integrated Circuits (ASICs), for example. Additionally, the method of the present invention may be performed by any combination of programmed general purpose computer components and/or custom hardware components. Therefore, nothing disclosed herein should be construed as limiting the present invention to a particular embodiment wherein the recited steps are performed by a specific combination of hardware components.

Figure 2 is a block diagram of a high level architectural view of selective expansion of HDL macros according to one embodiment of the present invention. In this example, a scan insertion tool includes a tokenizer module 110, an HDL parser module 140, a scan insertion module 145, and a file output module 150. The tokenizer module includes a module for reading HDL statements 215, a module for expanding HDL macros 220, and a module for selectively reading tokens 225.

Initially, an HDL file 205 is read by the tokenizer module 210. Briefly, the tokenizer module 210 reads an HDL line 215 from the HDL file 205 and stores lines as necessary in a lookahead list 230. This lookahead list is used to temporarily store lines as necessary. For example, several lines may need to be stored when a single HDL statement consists of more than one line of code. The tokenizer module 210 recursively expands HDL macros 220 and stores the expanded macros in a macro expansion list 235. As necessary, the tokenizer module 210 reads 225 tokens first from the lookahead list 230, then from the macro expansion list 235, and finally

from the HDL file 205 and passes them to the HDL parser module 240. The processing of the tokenizer module 210 will be discussed in greater detail below with reference to figure 3.

The HDL parser module 240 is presented with a stream of tokens from the tokenizer module "as if" all the macros have been expanded. This allows the parsing to be separated from the problem of selective macro expansion. The parsing program therefore sees only the expanded macros. The "as if" parsing process is discussed in greater detail below with reference to figure 4.

After the "as if" parsing process, the HDL parser module 240 then passes the parsed tokens to the scan insertion module 245 that inserts the scan commands. The file output module 250 is presented with a stream of tokens as if the scan macros have been expanded and non-scan related macros have not been expanded. Consequently, as will be described further below, the file output module 250 and the HDL parser module 240 may have different perspectives of the token stream, thereby providing an appropriate view of the token stream based on the context of its usage. The file output module 250 then writes the scan inserted HDL into a file 255. Using this method of scan insertion employing context sensitive macro expansion, a software tool can insert scan commands into an HDL design file in a way that preserves the text of the original file.

Figure 3 is a flowchart of tokenizer module processing according to one embodiment of the present invention. As mentioned above, the tokenizer module 210 first reads a line from the input file at processing block 305. If tokens are found in the input file at decision block 310, a determination is made in decision blocks 315 and 325 regarding from where the next token should be read. If at decision block 315 tokens are found in the lookahead buffer, the next token will be read from this buffer at processing block 320. Otherwise, if at decision block 325 tokens

are found in the expanded macro list, the next token will be read from this list at processing block 330. If no tokens are found in the lookahead buffer or the expanded macro list at decision blocks 315 and 325 respectively, the next token will be taken from the input file at processing block 335. The next token is then passed to the "as if" parser processing at processing block 340.

Figure 4 is a flowchart of "as if" parser processing according to one embodiment of the present invention. Initially, a determination is made at decision block 405 whether the token is a macro name. If the token is a macro name, the token is recursively expanded and added to the expanded macro list at processing block 410. As explained above, the tokenizer module 210 created multifaceted tokens that cause the parser module 240 to see an expanded token stream representing scan related macros and cause non-scan related expansions to be transparent to the file output module 250.

Further decisions are made at decision blocks 415, 425 and 435 based on the type of token found. If the token is found to be an end-of-line character at decision block 415, the input line of tokens is processed at processing block 420. If the token is found to be anything other than a symbol at decision block 425, the token is added to the current line token list at processing block 430. If the token is a symbol and at decision block 435 is determined to be a symbol that begins a macro definition, the macro name and definition are recorded and the token is added to the lookahead buffer at processing block 440. If the symbol does not begin a macro definition, the token is added to the current line token list at processing block 445.

By using a tokenizing module in conjunction with an "as if" parser as described above, the lower level complexity is separated from the parsing. The HDL parsing program is presented with a stream of tokens by the tokenizing module "as if" all the macros have been expanded. This allows the parsing to be separated from the problem of selective macro expansion. The parsing program sees only the expanded macros.

According to one embodiment of the present invention, to accomplish separating tokenizing and parsing, the HDL parser tokens are made multifaceted. That is, they could appear differently to different parts of the program. To accomplish this, additional fields are added to the parser token objects to contain the various views that different parts of the program see. With this data structure, the HDL parser can see the view of the token stream that show the expanded macros. However, the HDL output module can see a view of the token stream that limits expanded macros to those that involve scan related changes thereby presenting a view of the scan inserted HDL file that looks similar to the original HDL design file.

This multi view approach allows for fully expanding the HDL macros, but only writing out the portions that involve scan related changes thus preserving as much as possible the appearance of the original HDL design file. Tokens can also be "hidden" from the HDL parser by turning them into white space for the HDL parser. These hidden tokens are still present in the token stream, only hidden from the HDL parser. Therefore, if a macro requires no scan related changes and does not need to be expanded, the original macro is preserved and is available to be written out.

Figure 5 is a block diagram of a token object according to one embodiment of the present invention. This token object 505 includes a token type 510, a token string 515, an "IsScan" variable 520, a visibility variable 525, a macro argument index 530, a hidden token type 535, and a hidden token string 540. In this example, the token type 510 can be symbol, whitespace, operator, or quote among others. The token string 515 is the actual text of the token. The "isScan" variable 520 represents whether this token is related to scan insertion. The visibility variable 525 represents which parts of the program may view the token such as the parser, the output module, or others. The macro argument index 530 is used for expanding HDL macros.

The hidden token type 535 is the type hidden from the parser. The hidden token string 440 is the text of the token hidden from the parser.

Figure 6 is a block diagram illustrating an example of the use of multifaceted tokens according to one embodiment of the present invention. In this example, assume that an HDL file contains the macro definition statement DEFINE ASSIGN_X(x)=[y:=x]. The HDL statement ASSIGN_X(scanin) would be parsed and produce a token list as illustrated in Figure 6. First, the ASSIGN_X would be recognized as a macro call 605 since the DEFINE statement would place it in a symbol table. The macro call tokens "(", "scanin", and ")" would be added to a macro call token list 610-620. The macro call would be expanded to the token list "y", ":=", and "scanin" 625-635 with the "x" formal argument replaced by the "scanin" actual argument 635.

Notice that the macro call is still present in the first four tokens 605-620 but has been turned into white space as far as the parser is concerned. That is, the parser does not see the hidden token strings. The parser sees the expanded form of the macro call and will process those tokens 625-645. If the expanded macro involves scan, then the expanded macro tokens 625-645 will be used to write the updated line to the HDL file. If the expanded macro does not involve scan, then the original macro call 605-620 and 640-645 will be written to the HDL file thereby preserving the original HDL line.

Additionally, for the sake of example, assume that the HDL file contained the definition statement DEFINE CUBE_X(x) = [x*x*x]. The HDL statement ycubed := CUBE_X(y); would be parsed and produce a token list as indicated in blocks 650-670. Since the macro does not relate to scan insertion, there are no hidden tokens and the macro is not expanded.

Figure 7 is a block diagram illustrating results of tokenization as seen by an as if parser and an output module according to one embodiment of the present invention. In this example, an input file 705 contains the definition statements DEFINE ASSIGN_X(x) = [y:=x] and DEFINE

CUBE_X(x) = [x*x*x]. Later in the input file 705 the statements ASSIGN_X(SCANIN); and ycubed := CUBE_X(y); are found. That is, the statement relating the ASSIGN_X macro is scan related, the statement relating to the CUBE_X macro is not scan related. As a result, the parser will see a token stream 710 containing the expanded form of the ASSIGN_X macro and an unexpanded form of the CUBE_X macro. The output module will see a token stream 715 containing unexpanded forms of both macros.

Figure 8 is a flow diagram illustrating file output module processing according to one embodiment of the present invention. In this example, a token is received from the scan insertion module at processing block 805. At decision block 810, a determination is made whether the token involves scan related changes. If the token is not scan related it is written to the output file at processing block 815. If the token is scan related processing jumps to block 820. At processing block 820, a determination is made whether more tokens are to be received from the scan insertion module. Processing continues in this manner until no tokens remain.

DOCUMENT: 42390P9571